**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | A good practice is to validate input from all sources, especially untrusted sources. Using good input validation can cure a lot of common software vulnerabilities. One thing to remember is that it’s okay to be cautious of any outside sources. |
| 1. Heed Compiler Warnings | You should look to compile your code at the highest possible warning level available for your compiler. Then if you receive any warnings, you can get rid of them by modifying the code. You can also use specific analysis tools to find and delete other security risks. |
| 1. Architect and Design for Security Policies | You need to find a way to create a software architecture and then be able to instill and enforce security policies how you see fit. For example, with different things such as privileges at different times, or need to know/only give least access needed. |
| 1. Keep It Simple | In any case, you want to make design as simple, yet proficient as possible. When designs become too complex and convoluted; they become exponentially more difficult to manage and there are numerous other ways to intrude. |
| 1. Default Deny | By default, all access decisions are automatically denied, then there are certain set of circumstances in which access is granted. |
| 1. Adhere to the Principle of Least Privilege | I kind of alluded to this earlier a bit too, but all processes should execute with the least amount of privileges necessary. This allows for someone to not have too much power hopefully, or stumble upon data that they shouldn’t. |
| 1. Sanitize Data Sent to Other Systems | All data that is passed from systems to other systems should be sanitized or scrubbed. If not, potential attackers could use these things for injection attacks. |
| 1. Practice Defense in Depth | You should always manage security risk with multiple defensive strategies and layers with fail-overs. This way if one level fails or if an attacker gets past a layer, there is another layer there to pick up the attack. With multiple layers hopefully the attack can be responded to before anything becomes compromised. |
| 1. Use Effective Quality Assurance Techniques | QA is extremely important because it can be used for finding and getting rid of vulnerabilities. Pen testing and source code audits are a couple examples of effective QA techniques. Sometimes it is best to have external reviews done for unbiased opinions on systems. |
| 1. Adopt a Secure Coding Standard | You should look to adopt this standard to your target development language(s) and platform(s). |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Data Type Validation** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Need to validate the input for a data type so that the correct types of data are entered. For example, when asking for an input of an integer the user inputs an integer, when asking for a char the user enters a char, etc. |

| **Noncompliant Code** |
| --- |
| In this example there is no error checking for the data type: int |
| Int age;  Cout << “Enter your age: ” << endl;  Cin >> age >> endl;  /\* there is no error checking here. If the user enters a character  or string it could break the program \*/ |

| **Compliant Code** |
| --- |
| Code exists for error checking of the type of input. The while loop runs until valid input is entered. |
| Int age;  Cout << “Enter your age: ” << endl;  Cin >> age >> endl;  Typeid(age).name();  If( age != integer)  {  While(age != integer){  Cout << “please enter a number” << endl;  Cin >> age >> endl;  Typeid(age).name();  }  Cout << “Your age is “ << age << endl;  } |
|  |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Data Type Validation - This principle maps to this standard by enabling the user to validate a data type within their software. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium - High | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact data type validation. |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact data type validation |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Data Value Validation** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Similarly to the Data Type we need to provide some form of checking to make sure that the value fits within the parameters given. For example, if the string cannot be over 10 characters but the user inputs 11 characters. |

| **Noncompliant Code** |
| --- |
| This example allows a user to enter a person’s age but does not validate the integer data. That is, the user could enter -1 or 0 or a number too large to be represented by an integer data type. |
| Int age;  Cout << “Enter your age: ” << endl;  Cin >> age >> endl; |

| **Compliant Code** |
| --- |
| This example validates the input is a positive number. |
| Int age;  Cout << “Enter your age: ” << endl;  Cin >> age >> endl;  If(age<=0)  {  Cout << “Age cannot be negative or 0. Enter a Valid age” << endl;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Data Value Validation Principle – This principle validates if a data value fits within the necessary requirements. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium - High | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact data value validation. |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact data value validation |

#### Coding Standard 3

| **Coding Standard** | **Label** | **String Data Validation** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Validate the number of characters entered for a string. |

| **Noncompliant Code** |
| --- |
| The following code allows the user to enter the name of the car part but does not validate the data entered. |
| String name;  Cin >> name >> endl;  Cout << “Enter the part name: “ << name << endl; |

| **Compliant Code** |
| --- |
| The following code allows the user to enter the name of the car part and validates the part name to be 14 characters or less. |
| String name;  Cin >> name >> endl;  Cout << “Enter the part name: “ << name << endl;  Int name\_Length;  Name.length() = name\_Length;  If(Name\_length > 15)  {  While(name\_length > 15)  {  Cout << “Part name too long. Enter the part name again: ” << endl;  Cin >> name >> endl;  }  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** String Correctness Validation Principle – This principle is designed to validate string correctness within an application. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact string correctness validation. |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact string correctness validation |

#### Coding Standard 4

| **Coding Standard** | **Label** | **SQL Injection Validation** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Checks the user inputs for the string “or” for potential SQL Injection attacks and then will notify and stop subsequent queries. |

| **Noncompliant Code** |
| --- |
| This code accepts customer name and account number into the same variable. It is a perfect target for a SQL injection attack. |
| String name;  Cout << “Enter Customer name and Account Number: “ << endl;  Cin >> name >> endl; |

| **Compliant Code** |
| --- |
| This code detects the string “or” embedded within the user input for a possible SQL Injection. |
| if (sql.find("or") != std::string::npos) {  std::string cleaned = sql.c\_str();  cleaned = replace\_substring(cleaned, "\'", "\"");  std::cout << "Detected 'OR' in query! Possible SQL Injection!" << std::endl;  std::cout << "Stopped Query: " << sql.c\_str() << std::endl;  std::cout << "Escaped/Cleaned version of query: " << cleaned << std::endl;  return false;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] SQL Injection Validation Principle – This principle is designed to notify of SQL Injection possibilities and stop queries. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact SQL Injection notifcation |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact SQL Injection notification |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Overwrite Validation** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Validate length of character data entered by user. Malicious code can be written within an array of characters. |

| **Noncompliant Code** |
| --- |
| This code allows a user to enter a name but is limited to 20 characters. No code exists to validate the number of characters entered. The user could enter malicious data as the name. |
| Char name[20];  Cout << “Enter your name: “ << endl;  Cin >> name >> endl; |

| **Compliant Code** |
| --- |
| Defines the known integer values in memory before and after the array of character variable. After the name is entered the integer values are validated to make sure they are not changed. This limits the character string to be 20 characters or less. |
| Int value1;  Char name[20];  Int value2;  Value1 = -1;  Value2= -1;  Cout << “Enter your name: “ << endl;  Cin >> name >> endl;  If(value1 != -1)  {  //value1 has been overwritten  Cout << “Invalid characters entered for name.” << endl;  }  If(value2 != -1)  {  //value2 has been overwritten  Cout << “Invalid characters entered for name.” << endl;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] Memory Protection Validation Principle – This principle is designed to help guard against changes in memory when there shouldn’t be. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact memory changing protection |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact memory changing protection |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Data Assertion Validation** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Validating parameter values passed to a routine. |

| **Noncompliant Code** |
| --- |
| Calculated gross pay without validating pay\_rate or number of hours worked. |
| Double Grosspay(int hours, double pay\_Rate);  {  Double gross;  gross = hours \* pay\_Rate;  Return gross;  } |

| **Compliant Code** |
| --- |
| Calculated gross pay with assertion code for pay rate and hours worked. |
| Double Grosspay(int hours, double pay\_Rate);  {  Double gross;  If(pay\_Rate > 5.0)  {  Cout << “Invalid pay rate.” << endl;  }  If(hours > 40)  {  Cout << “Invalid hour total.” << endl;  }  gross = hours \* pay\_Rate;  Return gross;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** [Name the principle and explain how it maps to this standard.] Data Assertion Validation Principle – This principle is to assert that data entered is valid. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact assertion data |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact assertion data |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Floating Point Overflow Exception Validation** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Computes and checks for floating point overflow. When adding together multiple doubles, there is a risk of overflow. |

| **Noncompliant Code** |
| --- |
| Routine computes the sum of 3 doubles but no overflow exception is checked. |
| Double Comp\_sum(double value1, double value2, double value3);  {  Double sum;  Sum = value1 +value2 + value3;  Return sum;  } |

| **Compliant Code** |
| --- |
| Routine computes the sum of 3 doubles and checks for floating point overflow exception. |
| Double Comp\_sum(double value1, double value2, double value3);  {  Double sum;  Sum = value1 + value2 + value3;  if(fetestexcept([FE\_OVERFLOW](http://en.cppreference.com/w/cpp/numeric/fenv/FE_exceptions))) {  [cout](http://en.cppreference.com/w/cpp/io/cout) << "Floating Point Overflow detected" << endl;  return 0.0;  }  Return sum;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Exception Validation Principle – This principle is to validate that exceptions are handled correctly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might impact exception handling |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for any errors and warnings during output that might impact exception handling |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Integer Divide By Zero** |
| --- | --- | --- |
| Divide by Zero | [STD-008-CPP] | When two integers are divided, the denominator cannot be zero. |

| **Noncompliant Code** |
| --- |
| This code divides two integers but does not error check if num2 is equal to zero. |
| Int num1, num2;  Int result;  Result = num1/num2; |

| **Compliant Code** |
| --- |
| This code divides two integers and uses try and catch to make sure that you do not divide by zero. |
| Int num1, num2;  Int result;  try  {  if( num2 == 0 )  throw "Divide by zero error!";  else  Result = num1/num2;  cout << result <<endl;  }  catch (const char\* e)  {  cerr << e << endl;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Divide By Zero Validation Principle – This principle is to validate that a user is not dividing by zero within their application. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that might cause errors of divide by 0 |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for all errors listed within the file to negate any issues that might cause errors of divide by 0 |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Data Conversion Validation** |
| --- | --- | --- |
| Data Type Casting | [STD-009-CPP] | Avoid integer data casting when types differ. |

| **Noncompliant Code** |
| --- |
| Casting integer values from larger type to smaller type causes compiler warnings. Don’t resolve the warning with a variable cast. This code takes a long integer parameter and cast it to a int. |
| Long Sum(long num)  {  Int var1;  Int result;  Result = (int)num + 100;  Return result;  } |

| **Compliant Code** |
| --- |
| This code redefines the routine so that only ints are used and it eliminates the casting from long to int. |
| int Sum(int num)  {  Int var1;  Int result;  Result = num + 100;  Return result;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Data Type Casting Validation Principle – This principle is to avoid data type casting within applications. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that cause errors within data type casting |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for all errors listed within the file to negate any issues that cause errors within data type casting |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Long Integer Overflow** |
| --- | --- | --- |
| Integer Overflow | [STD-010-CPP] | When multiplying two long integers the maximum value is LONG\_MAX. The result can’t be compared to LONG\_MAX after the multiplication if it overflows. |

| **Noncompliant Code** |
| --- |
| This code multiplies two long integers without checking for integer overflow. |
| Long num1, num2;  Long result;  Result = num1\*num2; |

| **Compliant Code** |
| --- |
| This code multiplies two long integers after checking for integer overflow. |
| Long num1, num2;  Long result;  If(num1 > LONG\_MAX/num2)  {Cout << “Long Int Overflow detected.” << endl;  }else  Result = num1\*num2; |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):** Long Integer Overflow Validation Principle – This principle is to protect against long integer overflow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | Latest Release | CppCheck Checker | Check for all errors listed within the file to negate any issues that cause integer overflow within an application |
| Visual Studio (IDE) | Latest Release | Building of the program | Check for all errors listed within the file to negate any issues that cause integer overflow within an application |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Likely | Medium - High | High | 4 |
| STD-003-CPP | High | Likely | High | High | 5 |
| STD-004-CPP | High | Likely | High | High | 5 |
| STD-005-CPP | Medium | Medium | Medium | Medium | 3 |
| STD-006-CPP | Low | Low | Low | Low | 1 |
| STD-007-CPP | Low | Low | Low | Low | 1 |
| STD-008-CPP | Low | Low | Low | Low | 1 |
| STD-009-CPP | Low | Low | Low | Low | 1 |
| STD-010-CPP | Medium | Medium | Medium | Medium | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is holds the data in a protected state while it is being stored in a secure place waiting to be used. An encryption key is used to encrypt and decrypt the data as it is stored and leaves the storage. |
| Encryption at flight | Encryption at flight is the process of data being encrypted while it is being transferred from point to point. The data gets encrypted while being sent and then is sent along with a key to decrypt it when it gets to its destination. |
| Encryption in use | Encryption in use is when an application is being used the data is being encrypted. This allows the data to remain safe while the application is in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of the verifying a user’s identity in order to connect to a particular service. Services such as two factor authentication offer additional security to protect against unwanted users connecting. |
| Authorization | Authorization is the process of validating the user in question against the system in question. Authorization also determines the access and whether or not a user can complete the task in question. |
| Accounting | Accounting is often referred to the amount of resources that a user uses during their particular sessions. Accounting can be boiled down to any set of statistics and/or usage information. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/03/2022 | 10 Principles and Standards | Michael Jones |  |
| 3.0 | 04/09/2022 | Risk Assessment, Automated Detection, Automation, Summary of Risk Assessments, Policies of the 3 types of encryption and the 3 elements of the Triple-A framework | [Insert text.] |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |